3D Hand Positions (3DHP)

by

David William Krnavek

A Project Report Submitted in Partial Fulfillment
of the Requirements for the

Master's Degree in Computer Science

Approved, Graduate Advisory
Committee:

_____
Dr. Glenn G. Chappell, Chair

_____
Dr. Orion Lawlor

_____
Dr. Mitchell Roth

University of Alaska, Fairbanks
April, 2008

Table of Contents

1. Introduction

Computer applications traditionally utilize two dimensions for both input and output. Graphical displays use a two-dimensional array of pixels to represent information. Interaction with these systems is usually done with a mouse, which lets the user move a cursor around a two-dimensional plane.

Data, however, is not always easily described using only two dimensions. Real-world data tends to occur in a three-dimensional space. Geo-spatial information could be identified by its two-dimensional longitude and latitude, but in the real world, objects have some associated depth. Nearly all of the physical sciences deal with at least some information and data that occurs in three dimensions.

This increase in dimensionality leads to some difficulties in when trying to visualize this multidimensional data. 3D graphics are today a standard way of displaying data and information. Visualizations tend to require user interaction with the data – for instance, a permafrost researcher may want to zoom in on a certain region of his data. This interaction may be facilitated with traditional 2D input devices; however, interacting with the data may be better facilitated with a 3D form of input.

One laboratory concerned with creating and displaying such visualizations is the Discovery Lab at the University of Alaska Fairbanks. The Discovery lab is the main laboratory for running and creating virtual reality applications and environments for the

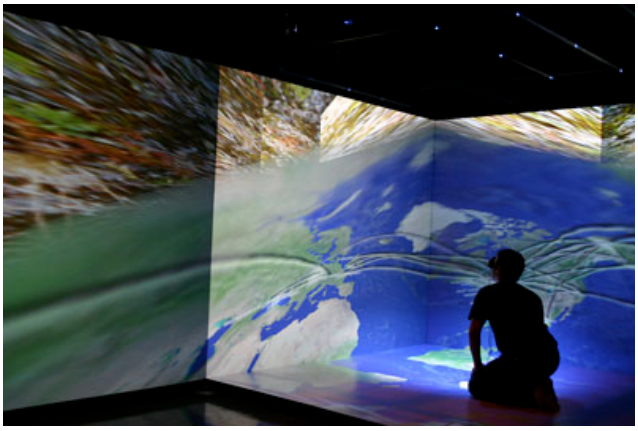


Figure 1.1 – ARSC Discovery Lab

Arctic Region Supercomputing Center (ARSC). The Discovery lab houses a Mechdyne Flying Flex CAVE environment, which consists of four large (10' by 7.5') projection surfaces. These surfaces are used to display immersive 3D environments and visualizations, such as the one in Figure 1.1.

The main virtual reality input device in the Discovery lab is an Intersense IS-900 tracking system. This system utilizes a handheld wand with an ultrasound microphone, such as the one shown in Figure 1.2a. A speaker array emits an ultrasonic chirp, which the microphone in the wand detects. A pair of stereo glasses is also fitted with a similar tracker that reports the head position and direction of the user (Figure 1.2b). This system is highly accurate (to within a fraction of a millimeter), but this leads to it also being quite expensive and out of the budget of most individuals.

Figure 1.2 – (a) Handheld wand and (b) head tracker

One of the Discovery lab's purposes, in addition to research, is outreach for the Fairbanks community and students at both the university and area schools. This outreach includes various tours of the lab and demonstrations of selected virtual reality applications. ARSC encourages the individuals in these tour groups to try out the various applications, in an effort to involve them directly in the virtual reality experience. However, the current system encourages only a single user at a given time, as there is

only a single wand and head tracker combination. An input system that could handle multiple simultaneous users is highly desired.

ARSC, the principal stakeholder in this project, has expressed interest in supplementing their current tracking system with another type of tracking system, using cameras and software to determine 3D user information. This type of system would be fairly inexpensive compared to the current system and would allow for different types of applications to be built, utilizing multiple 3D positions for input.

This project aims to create this camera-based 3D input device, called 3-Dimensional Hand Positions, or 3DHP. The following sections outline the stereo transformation process used to convert images from a pair of cameras into 3D hand positions, the design and implementation of the system, and finally the results and conclusions of this project.

2. Stereo Correspondence

Depth of vision refers to the ability of humans to gather distance information when viewing objects. Since the eyes are located in different locations on the head, each eye sees a slightly different image than the other. This slight difference between images, referred to as parallax, allows the brain to perceive the depth of the various objects within view.

The same basic principle can be applied to computer vision applications. Given two slightly differently positioned cameras, we will get a pair of similar, yet slightly different images. If we are able to locate a specific point that is visible in each image, we

can use the object's location in each image along with the known locations of the cameras to calculate the actual 3D location of the point.

A general model of a camera system is given in Figure 2.1. In this figure, we are concerned with three different coordinate systems. Digital cameras focus light through a lens into an image sensor. From there, it is sent to a CCD chip, where the image is broken into a finite number of pixels that, taken together, compose an image. The rows and columns of pixels of these images define the image coordinate system (ICS). A point (or pixel) in the ICS is represented by a row and column coordinate.

A camera coordinate system (CCS) is related to its respective ICS. The x-axis of the CCS is parallel to the rows of the ICS, and the y-axis is parallel to the columns of the ICS. Therefore, the z-axis is perpendicular to the row/column image plane. The focal length of the camera determines the distance from the origin of the CCS to the center of the ICS along the z-axis.

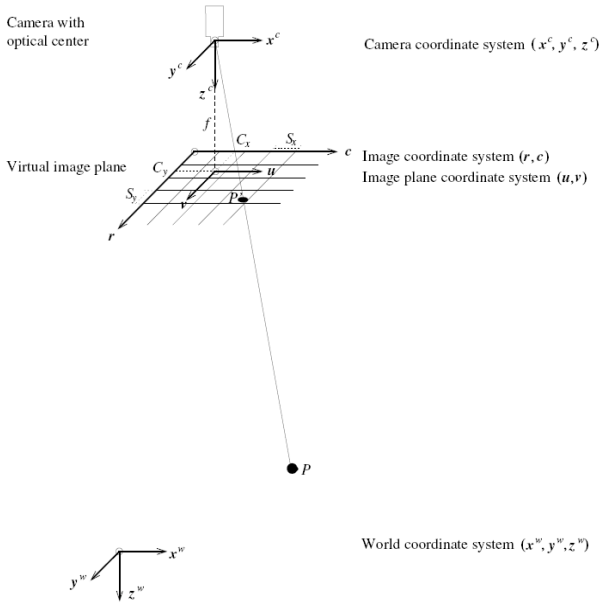


Figure 2.1 – General Camera System

A third coordinate system, the world coordinate system (WCS) can be any arbitrary 3D coordinate system. For computer vision applications such as motion capture, the origin of the WCS would more than likely be located in the middle of a stage area with the x-y plane along the floor of the room.

Figure 2.2 represents a model of a general stereo camera system. A specific point within view of a camera will be reflected through the camera's lens, into the image sensor, and onto the camera's CCD chip, where it is stored as an array of pixels. The location of the point P appears in each camera's image plane as point P' and is referred to as a conjugate point. We can trace a line from the origin of each camera's CCS through each conjugate point into infinity. The intersection of these lines gives the location of point P in the WCS.

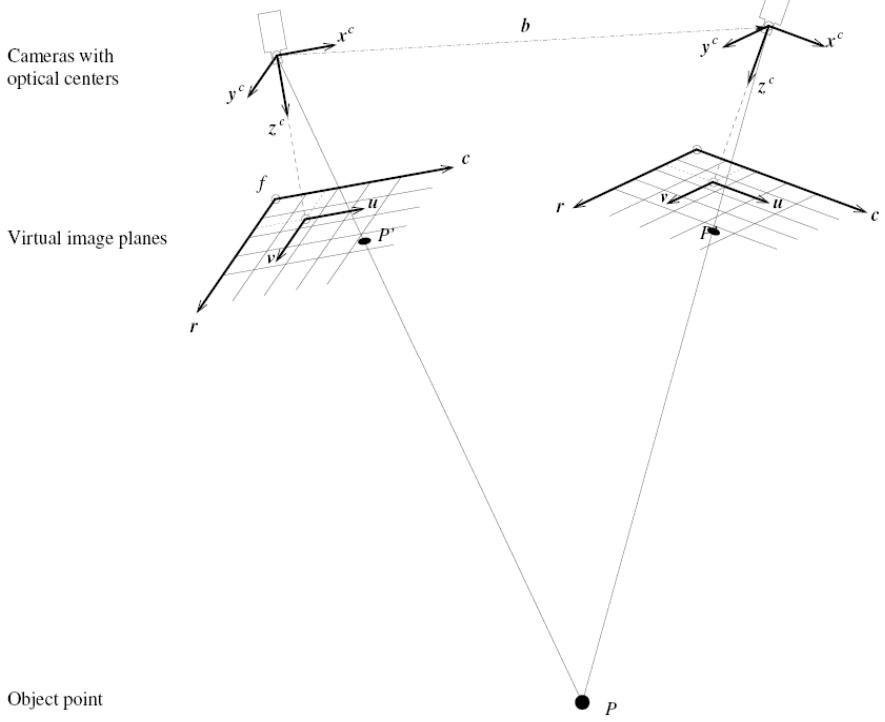


Figure 2.2 – Stereo Camera System

Therefore, in order to transform a conjugate pair of image points to a 3D position, we need to know some information about the internal workings of the cameras. This information, known as the intrinsic camera parameters, is listed in Table 2.1.

In addition to this internal information about the cameras, we need to know how the cameras are positioned with respect to the world origin. This external information, known as the extrinsic camera parameters, simply consist of a translation vector specifying the location of the camera in relation to the world origin and a rotation vector specifying the camera's rotation with respect to the world coordinate system.

Intrinsic Camera Parameters
Focal length of the lens
Radial distortion coefficient
Width of a cell on the CCD-chip
Height of a cell on the CCD-chip
X-coordinate of the image center
Y-coordinate of the image center

Table 2.1 – Intrinsic Parameters

Camera calibration involves two forms of input: a list of known image coordinates for a list of known world coordinates, and a set of initial intrinsic parameters. The calibration process is essentially a minimization problem, where we project the known world points into image points, then adjust the parameters until we minimize the errors in distance.

A calibration plate is used to facilitate the mapping of world coordinates into image coordinates. The calibration plate consists of a grid of evenly spaced dots, such as the one in Figure 2.3. We can physically measure the locations of the dots, and the image coordinates for the dots can be found by searching an image for this known pattern. In order to maintain a high degree of accuracy, the calibration is carried out using a series of images, each with the calibration plate in a different pose. For each of these images, we determine the image coordinates of each dot, project their respective locations to calculate a world coordinate, and compare this

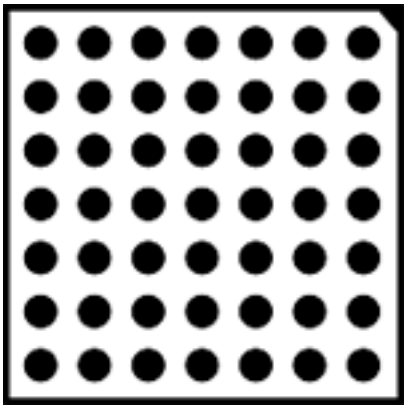


Figure 2.3 – Calibration Plate

value to the known coordinate. This process iteratively modifies the camera parameters until we minimize the error between calculated and known world coordinates.

This calibration process determines the intrinsic camera parameters by iteratively adjusting a set of initial values. The extrinsic parameters are determined using an image of the calibration plate placed on world plane defined by z=0. It is possible to determine the pose of the calibration plate by again projecting the image points into world coordinates, comparing these to the known world coordinates, and minimizing the errors.

3. Project Requirements

The requirements of the system must be determined before the system design and implementation can be carried out. The functional and non-functional requirements of the system are outlined below, along with other miscellaneous system constraints.

3.1 Functional Requirements

The functional requirements for this system are listed below. These requirements will determine the outward functionality of the system.

- The finished project is to be run from the command line without the need for a graphical user interface.
- The only required user interaction will be to start and stop the program and to toggle printing of information to the command window.
- This system shall not interfere in any way with the current tracking system in the Discovery lab.

- The system will simultaneously grab separate images from a pair of cameras connected to the system.
- To facilitate the location of the user's hands, the user will be wearing some sort of relatively easily detectable hand markers.
- The system will search for at least two different hand markers, corresponding to the users left and right hands.
- The incoming images will be searched for the hand markers. The row/column coordinate of the center of each hand marker will be determined for each image.
- The conjugate pair of image coordinates shall be transformed into 3D world coordinates.
- The found hand position shall be printed to the command window if desired.
- The found hand positions shall be formatted into a UDP packet and sent to a multicast address, in order to be used by multiple programs simultaneously.
- These packets will only be sent if a hand marker has been found.

3.2 Non-functional Requirements

In addition to the functional requirements of the system, several non-functional requirements are addressed. These requirements include quality constraints on the system. The non-functional requirements for this system are listed below, in descending order of importance.

1. Performance: For an input device to be usable, it must be responsive to user input. Therefore, the performance of this system is a top priority, as slow tracking speeds will render it practically useless for real-time applications.

Since this system is based on cameras, the maximum performance of the system is dictated by the camera's maximum frame rate.

2. Accuracy: Similarly, input devices need to be accurate to be usable. 3D hand positions are useless as a form of input if they do not reflect the user's actual hand positions. Therefore, the accuracy of the system should be a high priority.
3. Modifiability: Being able to add or replace certain features of the system is a high priority. Since this system is being designed and implemented in an ongoing research facility, the ability to easily modify the code is strongly desired. The system should be designed in a modular fashion so that various components can be changed and improved.
4. Reliability**:** The system should not crash unexpectedly, or it will become less and less desirable as a form of input. Therefore, this system must be able to run and output expected results as long as the user desires.
5. Interoperability: The system shall be installed alongside other equipment in the laboratory. It should in no way hinder the use of the existing lab hardware or software. In addition, it should be able to be run at the same time as the equipment in the lab as a complementary form of input.
6. Configurability: Changes in the physical layout of the system should be anticipated, such as moving the cameras to new locations. Methods or routines should be implemented to assist the user in recalibrating the system as desired. In addition, simple configuration changes in the system, such as what to search for or the destination of the output, should be implemented in

an external file. This way the system can be reconfigured without needing to recompile the source code.

3.3 Other Requirements

Some of the equipment needed for this project was previously purchased before any project requirements or design was specified. Therefore, this project also has the following hardware requirements:

1. Program to run on a PC with a Windows XP, Service Pack 2 operating system
2. System will use Sumix M71 USB 2.0 cameras

One constraint that was created by using the Sumix cameras was that the USB 2.0 cameras have a maximum cable length of 5 meters. Therefore, the placement of the computer used to run the 3DHP system was also confined to a certain area.

4. System Design

The main functionality of the 3DHP system consists of taking a pair of images, determining the user's 3D hand positions from these images, and broadcasting these hand positions to a multicast address. These three processing steps are separated into three functional modules: the Image Acquisition module, the Data Extraction module, and the Data Transport module. The responsibilities of each module are defined below. Besides these three modules, the 3DHP system also includes a calibration routine and a preferences file, which are also described below.

4.1 Image Acquisition Module

The 3DHP system utilizes two digital cameras. These cameras must be initialized for use in the system and the images from these cameras must be stored in a format that is accessible to the other modules in the system. In addition, for the sake of performance, we would like to have images ready for the modules whenever they are requested, instead of having to wait for new images. This functionality is delivered to the system by the Image Acquisition Module.

The camera initialization step is straightforward – each camera is set to the desired resolution and exposure time, and any color and white balance adjustments are made. After the cameras are properly set up for the system, they are given the proper signal to start grabbing images.

Grabbing an image from a camera doesn't happen instantaneously. The camera needs time to write the image data to its memory. We are emphasizing system performance, so any unnecessary waiting is undesirable. Therefore, a buffer will be used to store the most recently grabbed images from each camera. Other system modules will be able to take images from these buffers, and the cameras will write to the buffer whenever a new image is ready.

4.2 Data Extraction Module

The data extraction module contains the primary functionality of the 3DHP system – namely to calculate the 3D position of a user's hands given a pair of images. This can be split up into two steps. First we search the pair of images for the user's hand

positions. Then we can transform these image coordinate pairs into 3D positions using the stereo intersection method outlined in Section 2.

Locating a user's hands in an image can be a complex task. Techniques for segmenting the user's hands from a background might include searching for a specific contour relating to the shape of a user's hand or determining the hand positions from an outline of the user's pose. These methods are computationally expensive – which leads to slower performance – and require a considerable amount of training for their respective systems. Another method of finding hand positions involves searching for a specific color range pertaining to human skin. While researching appropriate methods, it was determined that the cameras used in this project are very sensitive to infrared light. This resulted in colors of both clothing and human skin being quite similar and therefore difficult to differentiate.

For this project, hand tracking is simplified by having the user hold a colored marker in each hand. The markers will be colored using light emitting diodes, which emit colors that aren't adversely affected by the infrared sensitivity of the cameras. Since the user will be holding distinct, colored markers, the hands should be easily distinguishable from the rest of the scene. This colored marker method of locating the users hands also allows for easy distinction between the user's left and right hands. Different colors can be used to differentiate between different hands, and adding new markers to search for is trivial.

This module takes new images from the image buffer described above. The incoming images are searched for a specific color (within a predetermined threshold), and if something is found, it is assumed to be the desired hand marker. If nothing is found,

we can assume that the hand marker isn't in view of the camera and start again with a new image. The module will not handle ambiguous information. If we find more than one object of the same color, we won't bother trying to resolve which object is the desired hand marker; we will simply start the search again with a new image. When we end up with a single object corresponding to the proper color range, we will determine the center of this object and pass it on to a stereo rectification function.

The second step involves the transformation of the image coordinate pairs into 3D world positions. This step requires calibration data for each of the cameras – specifically the calibration data specified in Table 2.1. Given a pair of image coordinates specifying the location of each hand marker and this camera calibration data, we can perform the stereo rectification procedure as outlined in Section 2.

The three-dimensional hand locations are then passed along to the Data Transport module to be sent out to the network. Then the module repeats the process of grabbing images from the buffer, searching for the hand positions, and transforming them into three-dimensional world coordinates.

4.3 Data Transport Module

One of the requirements of this system is to make the determined hand positions available to multiple programs at the same time. This is accomplished with the use of IP Multicast packets, which effectively broadcast the data to a specified IP address. The Data Transport module is concerned with taking the located 3D hand positions and broadcasting them to the network. This module will format the hand positions into IP

datagrams, which are sent to a multicast IP address, where they are available to multiple interested receivers on the network.

The Data Transport Module will provide the following functionality to the system:

1. Set up an outgoing multicast connection
2. Format each incoming marker's world coordinates into a character string with an identifier followed by the coordinates
3. Send this string to a specific multicast address

4.4 Calibration Routine

Before doing any stereo rectification, we will need to know information about the camera system. Specifically, for each camera we need to determine the intrinsic and extrinsic camera parameters outlined in Section 2.

The Calibration Routine shall take the following input: initial values for the intrinsic camera parameters, a list of calibration plate world coordinates, and a sequence of images of the calibration plate. Using this information, the calibration routine will apply the method outlined in Section 2 to determine the intrinsic and extrinsic camera parameters. These parameters will be used in the Data Extraction module to determine the world coordinates of the user's hand markers.

4.5 Preferences File

In order to increase the modifiability of the system, a preferences file will be used to specify a few system parameters. Instead of requiring the user to recompile the system

when changing either the color range used to find the hand markers or the network address for the hand positions, we can simply modify this file.

The Preferences File will provide the following information to the system:

- The color range used to search for the left-hand marker
- The color range used to search for the right-hand marker
- The multicast IP address and port that the marker positions are to be sent to

This file will be read and parsed by the system on initialization. The color information is passed to the Data Extraction module, and the multicast network information is passed to the Data Transport module.

4.6 System Architecture and Workflow

The modules defined above, taken together, define the 3DHP system. A view of the system architecture is given in Figure 4.1, along with a list of the functions and variables defined within each module.

The basic workflow of the 3DHP system is outlined below:

1. The Image Acquisition module initializes the cameras and starts taking and storing images in a buffer. This module continues taking and storing images indefinitely.
2. The Data Extraction Module takes the most recent images from the buffer and extracts the image locations of the markers. The specific colors of the markers to look for are taken from the Preferences File. Using the intrinsic and extrinsic camera parameters determined by the Calibration Routine, the 3D world coordinates of the markers are determined.

3. These world coordinates are passed to the Data Transport Module, which formats and send them as a packet to a multicast address. The multicast address and port are specified in the Preferences File.

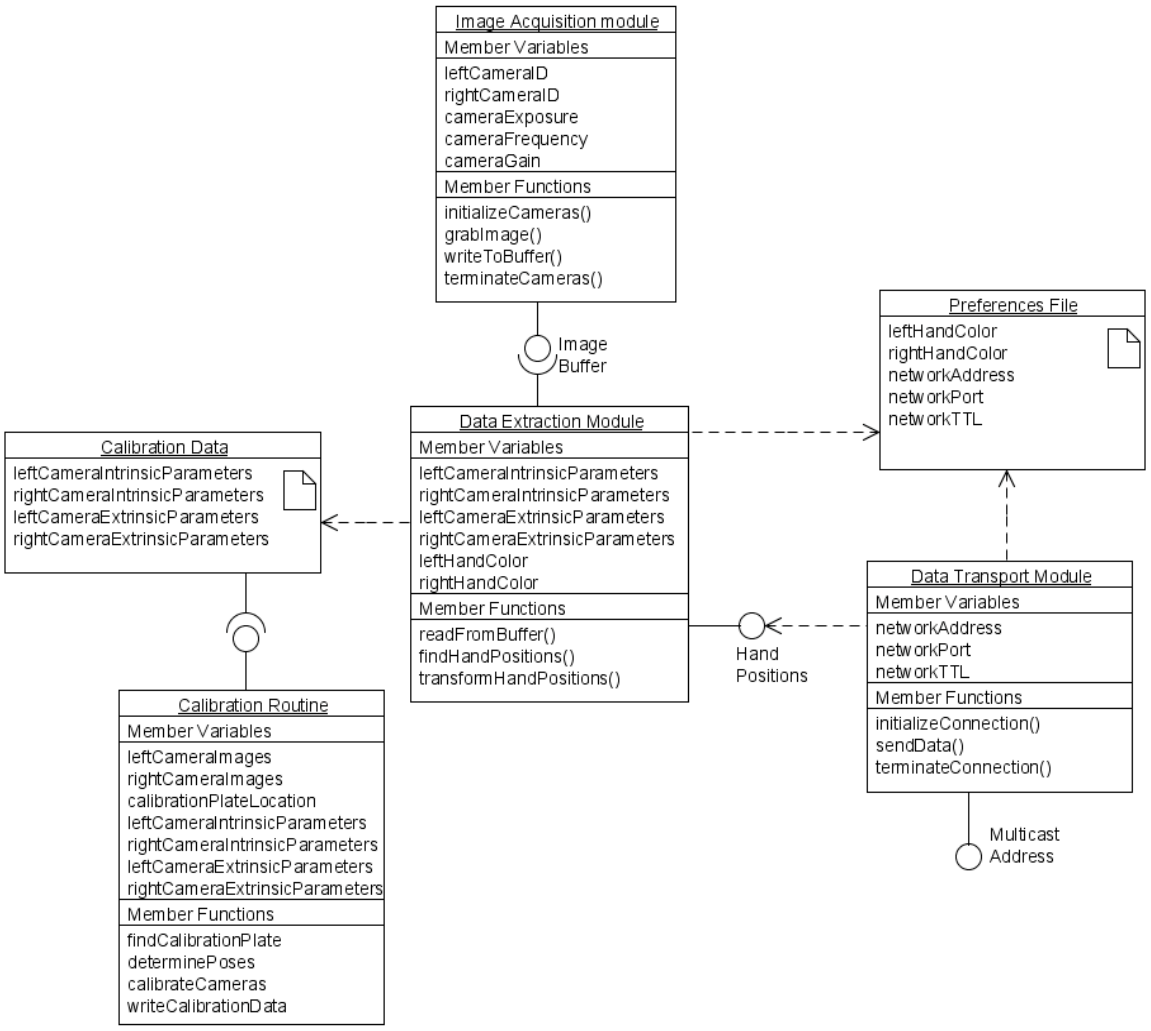


Figure 4.1 – Architectural View of 3DHP System

The Calibration Routine must be run before the system starts. This routine doesn't need to be run each time the system starts, as long as the calibration information is maintained in the proper files. In addition, before starting the system, the Preferences File must be filled with the required information. The system is started and runs until a

termination signal is given. Each module receives this termination signal and stops processing. Once all the modules are stopped, the system halts.

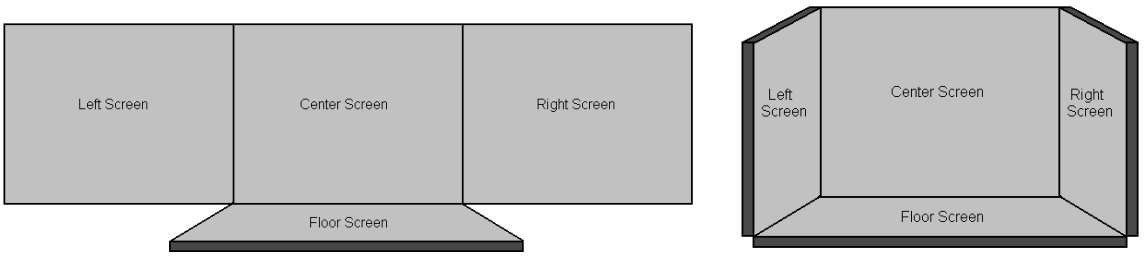
5. Implementation

With the design of the system in place, the implementation of the system is able to begin. This section addresses the different hardware components of the system, the software libraries used, and the various implementations of the system components.

5.1 System Hardware

The primary hardware devices in the 3DHP system are the cameras. The specific cameras used are a pair of Sumix M71 CMOS cameras with USB 2.0 interface. The M71 is a 1.3 megapixel camera with a maximum resolution of 1280 x 1024 pixels. These cameras allow for the individual control of the pixel gain, exposure time, and pixel rate. In addition, the output frames can be decimated by a factor of 2, 4, or 8, resulting in output resolutions of 640 x 512, 320 x 256, or 160 x 128, respectively.

The 3DHP system is to be integrated into a pre-existing CAVE environment. This environment consists of four individual projection surfaces – three vertical wall



(a) Powerwall Configuration (b) Closed Configuration

Figure 5.1 – CAVE Environment

displays and a floor display. The environment can be used in a "powerwall" configuration, or the two outer wall displays can be rotated inward, resulting in an immersive four-walled room. These two configurations are illustrated in Figure 5.1.

This type of environment nicely defines a staging area for users, namely the 10' x 10' floor display. The cameras used in the 3DHP system are placed in the upper corners of the center screen, as illustrated in Figure 5.2. They are then centered on a point approximately one meter above the center of the floor screen. This camera arrangement maximizes the visibility of a user standing on the floor screen. Graphical feedback may be displayed on the center screen, which will naturally orient the user facing the cameras.

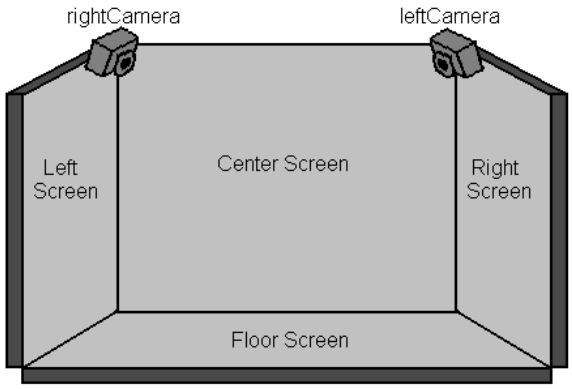


Figure 5.2 – Camera Locations in CAVE Environment

The cameras are connected via a USB 2.0 interface to a Windows XP machine, which is used to develop and run the 3DHP system. This machine is also connected to the Internet through a local network in order to send multicast packets.

The final hardware component used in the 3DHP system is the LED hand markers. These markers use the simple circuit shown in Figure 5.3. This circuit utilizes a 3V coin battery to power an LED. The resistor limits the current flowing into

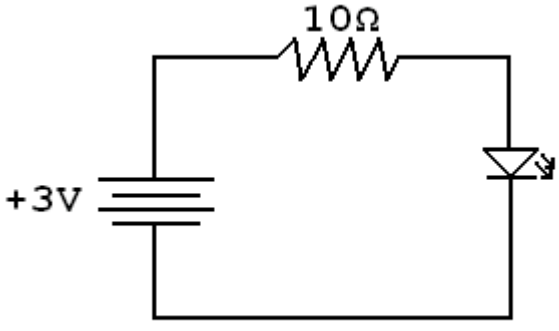


Figure 5.3 – LED Hand Marker Schematic

the LED. A larger resistor might be used in the circuit for a dimmer LED but longer battery life.

In order to maximize visibility at all angles, the LED in the circuit is fitted inside a standard ping-pong ball. The emitted light is diffused about the entire surface of the ping-pong ball, resulting in very slight intensity differences for varying angles.

The parts list for each LED hand marker is given in Table 5.1.

Part	Quantity	Description
5mm colored LED	1	Emits a colored light
3V coin battery cell	1	Supplies current to the LED
Coin cell battery holder	1	Holds the battery cell
White ping pong ball	1	Diffuses the LED's light
10□ Resistor	1	Limits the current flowing into the LED
Hookup wire	N/A	Used to make connections between components

Table 5.1 – LED Marker Parts List

Constructing the LED hand markers involves soldering the circuit together and fitting the LED inside the ping-pong ball. This entails drilling a small (5mm) hole inside the ping-pong ball and simply sliding the LED inside the hole. A rubber fitting is used to secure the LED inside the hole. This ensures a snug fit for the LED without making it permanent. The physical layout of an LED hand marker is shown in Figure 5.4.

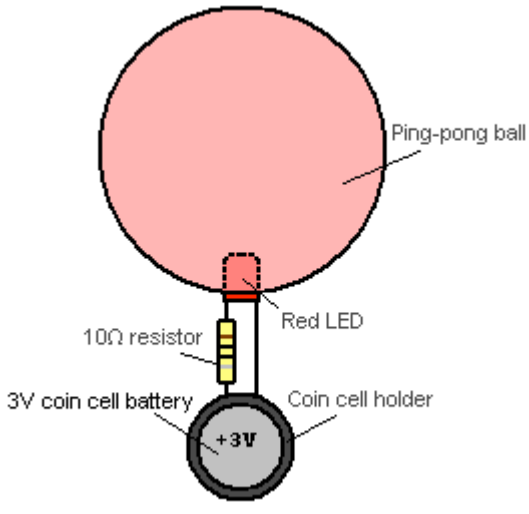


Figure 5.4 – Physical Layout of LED Hand Markers

Two different hand markers are made – one with a red LED and the other with a green LED, since they are both readily available and optically differentiable.

5.2 Software Libraries

The image processing functions of 3DHP are implemented using the HALCON library. HALCON is a commercial library that provides over 1300 machine vision functions and operations. HALCON library functions are used for the image analysis, the stereo rectification process, and the camera calibration.

The main functions used from this library are listed in Table 5.2, along with a brief description of its functionality. Any other library functions used are for creating and writing pose information to a file (such as in the Calibration Routine), or initializing the different HALCON objects.

Function	Description
read_pose()	Reads camera extrinsic parameters from an external file
read_cam_par()	Reads camera intrinsic parameters from an external file
gen_image_interleaved()	Stores interleaved image stored in memory as HALCON object
decompose3()	Splits multi-channel image into a set of single-channel images
threshold()	Selects pixel values in between a low and high value
intersection()	Finds the common pixels between two objects
connection()	Splits a thresholded image into connected components
select_shape()	Selects components based on shape and/or size
count_obj()	Counts the number of connected components in an object
area_center()	Determines the center (x, y) of an object
intersect_lines_of_sight()	Computes the 3D point of intersection between two lines of sight from different cameras
affine_trans_point_3d()	Applies an arbitrary 3D transformation to a 3D point
find_caltab()	Searches an image for the HALCON calibration object
find_marks_and_pose()	Extracts the individual calibration points from an image of the HALCON calibration object
binocular_calibration()	Determine the intrinsic and extrinsic camera parameters from a group of world and image coordinates

Table 5.2 – HALCON Functions

An advanced programming interface (API) comes with the Sumix M71 cameras used in the 3DHP system. This API contains functions for opening and closing the cameras, grabbing images, and setting other camera settings such as exposure, gain, and frequency. The API functions and data structures are specified in a C++ header file, with a dynamically linked library handling the implementation of the functions.

The Image Acquisition module is the only 3DHP module that uses the Sumix API. The functions used are given in Table 5.3, along with a brief description of each. These functions are used once for each camera, which receives its own device identifier from the CxOpenDevice function.

Function	Description
CxOpenDevice	Opens the camera device
CxCloseDevice	Closes the camera device
CxSetExposureMs	Sets the camera exposure value
CxSetFrequency	Sets the camera sensor frequency
CxSetGain	Sets the camera sensor gain
CxStartVideo	Starts the camera video stream
CxStopVideo	Stops the camera video stream
CxSetBayerAlg	Sets the Bayer convertion algorithm (monochrome, nearest neighbor, bilinear, laplacian, or Bayer average)
CxBayerToRgb	Converts the Bayer matrix to a 24-bit RGB color frame
CxGetFramePointer	Returns a pointer to the driver's frame memory buffer

Table 5.3 – Sumix API Functions

The other software library used in the 3DHP system is the Windows Sockets API version 2.0, also known as Winsock. Winsock provides the interface between the Microsoft Windows operating system and the TCP/IP stacks. The Winsock API provides functions for setting up and transmitting data across a network using sockets. Specifically, the Data Transport module of 3DHP uses Winsock to transmit the hand

positions to a multicast address via UDP datagrams. The socket setup and transmission is done with the help of functions provided by the Winsock API.

Finally, the Microsoft Windows kernel (kernel32.dll) is used to implement multiple execution threads. The kernel is responsible for forking the different execution threads and maintaining thread synchronization.

5.3 Implementation Specifics

The libraries described above all make use of C/C++ APIs. Therefore, the 3DHP system was implemented using C++. The different libraries also made use of dynamically linked libraries, which necessitated development in a Windows environment. The compiler used for development was Microsoft Visual C++ .NET.

The main functionality of the system is handled by the three primary system modules: the Image Acquisition module, Data Extraction module, and the Data Transport module. These modules are implemented together in a single command line program.

Two of the modules, the Image Acquisition module and the Data Extraction module, have their functionality implemented in two separate program execution threads. This allows for the modules to work independently, with images being grabbed and processed simultaneously.

These two modules can work independently thanks to the use of an image buffer. New images are stored in the buffer by the Image Acquisition module. By buffering incoming images, images are always available when requested by the Data Extraction module. The image buffer is implemented as a shared memory location that is accessible to the two program threads. In order to make sure that the buffer isn't being read by the

Data Extraction module while it is being written by the Image Acquisition module, a mutually exclusive variable (mutex) is used. Before accessing the shared buffer, the mutex is told to block any other threads from using the resource. After accessing the buffer, the mutex is told to stop blocking access.

The other primary module in the system, the Data Transport module, is implemented as a set of functions. One function sets up the network connection, while another is used to send data to the desired network address. When the Data Extraction module finds new hand positions, it simply calls this function, which sends the hand positions out to the network.

The calibration data and preferences file are stored on disk and are accessed at the start of the program. The desired data is stored in variables for use by the individual program threads. Finally, a third execution thread polls the keyboard for input. If an escape character is typed, the other program threads are told to quit and the program exits.

The main program operates with the following sequence:

1. Read in calibration data and store in global variables
2. Read system preferences data and store in global variables
3. Initialize the network connection (part of the Data Transport module)
4. Start the Image Acquisition thread
5. Start the Data Extraction thread
6. Start the keyboard thread

At this point, the individual modules operate together to grab images, search for hand positions, and output these positions to the desired network address. When the

keyboard thread detects an escape character, a global variable, “timeToQuit” is updated. The other threads check the status of this variable before starting again with their different procedures. If the variable is set, the individual threads start their different shutdown sequences. The entire program quits after all of the threads are finished with their respective shutdown sequences.

The other component of the system – the calibration routine – is implemented as a separate program. This calibration program takes a set of images with a visible calibration plate. From these shots of the calibration plate, the function determines the internal and external camera parameters for the two cameras in the system. These camera parameters are written in files, which are then accessible to the Data Extraction module in the 3DHP main program.

5.3.1 Image Acquisition Module Implementation

The Image Acquisition module is implemented in a straightforward fashion. Each camera is initialized as follows:

1. Open camera device
2. Set image resolution to 1280x1024
3. Set camera frequency to 24 MHz
4. Set camera exposure to 10 ms
5. Set camera gain to 100%
6. Start streaming images

Once the cameras are initialized, the function enters a loop that exits when the function receives the exit signal. Inside this infinite loop, the cameras are polled for new

images, which are saved in the shared memory location. While saving the new images, the appropriate mutexes are locked to prevent the images from being read by the Data Extraction Module. Once the images are moved into their memory locations, the mutexes are unlocked.

5.3.2 Data Extraction Module Implementation

The Data Extraction Module is responsible for searching the incoming images for the appropriate hand markers, and then transforming these image coordinates into 3D world coordinates. The first part of this functionality is implemented as the function select_object_by_color, which takes an image and color range to search for and returns the row and column of the center of the found object. This method is given below:

1. The incoming image is split into three grayscale images corresponding to its red, green, and blue channels
2. Each of these channels is thresholded between the input color ranges. This results in a binary image where a pixel is on if the corresponding pixel in the grayscale image is within the range.
3. The intersection of these three binary images is taken, which results in a binary image of all objects in the original image within the desired color range
4. The number of connected components is calculated (distinct non-touching objects in the binary image).
5. If there is only one object, the center of this object is returned in row, column coordinates. Otherwise, the function returns coordinates of –1, -1.

The previous function is run four separate times, twice to find the red and green markers in the left camera's image, and twice to find the red and green markers in the right camera's image. If the function found an object in both camera images, the respective image coordinates are sent to the HALCON function intersect_lines_of_sight. This function determines the intersection point from these image coordinates and the camera intrinsic and extrinsic parameters. These resulting 3D coordinates are based on the primary camera's origin, so they are transformed a final time to the world coordinate system using the HALCON function affine_trans_point_3d.

After this final transformation, we are left with either the red or green hand marker's 3D position. These positions are then passed to the Data Transport module.

5.3.3 Data Transport Module Implementation

The Data Transport module is implemented as two different functions. One function initializes the multicast datagram socket. The second function sends a given character string to the multicast address.

The socket initialization only needs to be run once – when the system starts. The preferences file specifies the multicast address, port, and time to live for the outgoing packets.

Once the socket connection is made, we can simply send character strings to the destination address. The strings are formatted in the following style, where xPos refers to the x-coordinate of the hand marker and yPos and zPos refer to the y-coordinates and z-coordinates, respectively.

"marker0 xPos yPos zPos"

The module can print a message to indicate whether or not the message was sent or if an error occurred. The printing of these messages will affect the overall speed of the system, so once it is seen that the system is running correctly, these messages can be disabled.

5.3.4 Calibration Routine Implementation

The Calibration Routine is implemented and run in a separate program using HALCON's hDevelop environment. This environment lets a user run a program created with different HALCON functions. It is mostly used for testing the different functions and it allows for the user to display the images and objects he is currently working with. In addition, the user can step through functions one at a time, as in a debug mode, which is handy for visualizing the results of the camera calibration process, such as whether or not the marks were all found.

The Calibration Routine requires a set of images of the calibration plate as seen through the cameras. The HALCON library includes a postscript file describing the layout of the calibration plate. A large (1 square meter) printout of this file is made and mounted on a piece of foam core. The calibration plate is positioned at a different angle in each image, across as much of the staging area as is possible. Two of these sample calibration images are shown in Figure 5.5 (an image from the left camera and right camera respectively), along with marks on the corners that the routine detected (which will be discussed later).

After loading the images into the program (this implementation uses 15 different orientations), the find_caltab function is called, which searches the image for the calibration plate. This resulting image is then passed to the find_marks_and_pose

function, which determines the image coordinates of the individual calibration marks in each image. If this function finds all of the points, they are passed into a list of all image points. This list, along with a file specifying the layout of the calibration marks and initial camera parameters, is sent to the binocular_calibration function, which determines the intrinsic and extrinsic camera parameters.

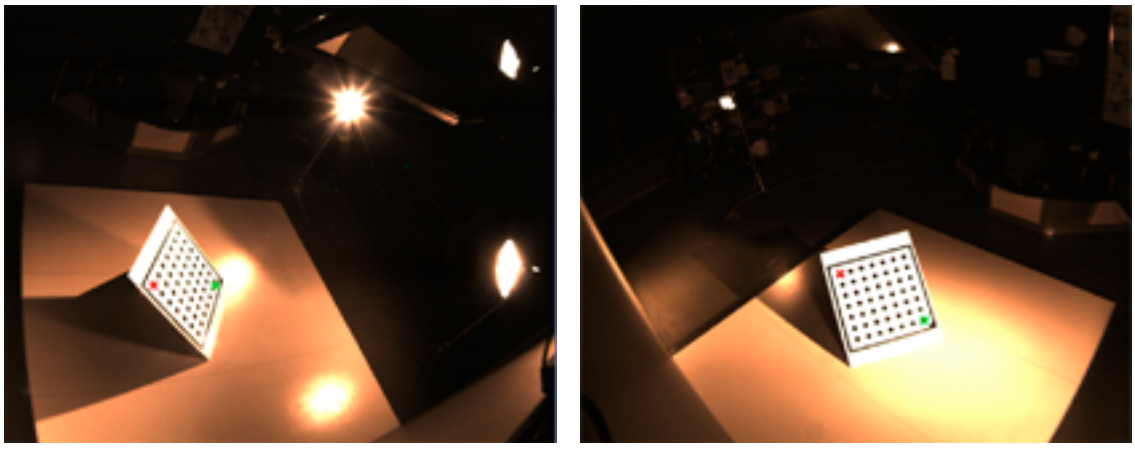


Figure 5.5 – Sample Calibration Routine Input

6. Results

6.1 System Performance

Several tests were carried out in an effort to determine if the 3DHP system met certain non-functional requirements. The most important non-functional requirement for this system, as outlined in Section 3.1, is performance. Two different tests were run to quantitatively measure the system's performance.

The first performance-related test program was built to find the number of marker positions that the system is able to locate every second. This program receives all messages sent to the 3DHP's specified multicast address and checks to see if the message

refers to either the red or green marker. Two counters, one for each marker, are incremented every time a new position is found for its respective marker. A timer is started when the first marker is found, and stopped when the hundredth marker is found. The number of marker positions located per second is easily determined by dividing 100 by this amount of time.

This test program is run while using the system in a typical fashion. The markers were both moved around the staging area, not simply set in one location. Figure 6.1 outlines the number of markers located per second.

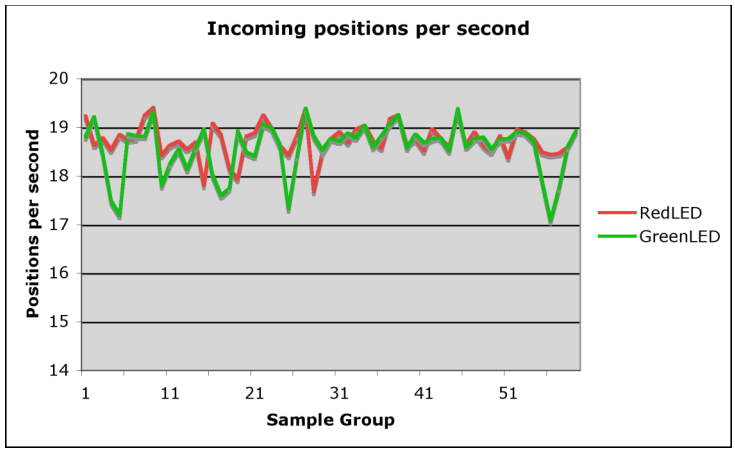


Figure 6.1 – Positions Per Second

The 3DHP system was then modified to effectively leave out the Data Extraction Module. This modified system grabs images in a normal fashion and outputs a nondescript value to the Data Transport module, which sends this irrelevant information to the multicast address. A receiving program is set up that calculates the number of raw messages received per second. This is used to measure the raw camera rate that can be expected with the current system. Figure 6.2 shows the number of images found in a sample run.

It can be seen that the Data Extraction module adds a fair bit of overhead to the system. The base rate that we can get by just reading from the cameras is on average 24.56 frames per second, while the average number of calculated hand marker positions is 18.62 per second. Some more optimization can be made to the system to achieve a

faster framerate, but it should be noted that the system is already running at a fairly respective 75.8% of the optimal framerate.

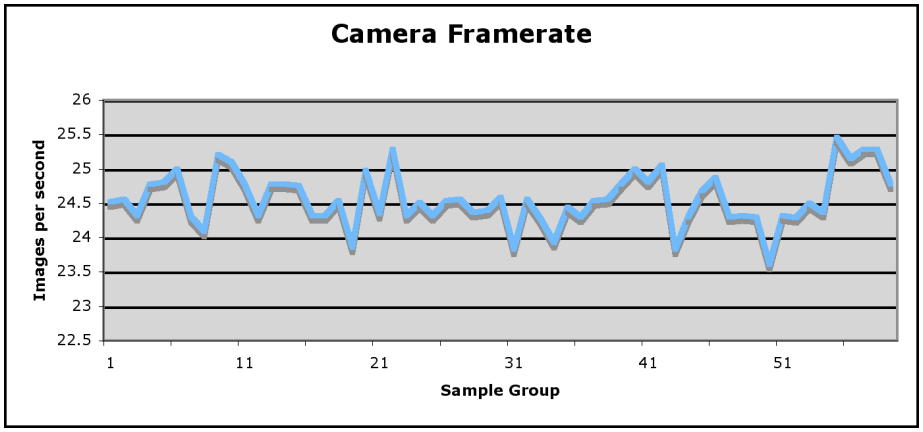


Figure 6.2 – Base Camera Framerate

A final test relating to system performance is concerned with measuring the percentage of markers located in a scene. The 3DHP system is modified to output messages based on whether or not it is able to locate a marker in the scene. When the system fails to locate a red marker in the frame, it outputs the message, "RedMarker 0." Conversely, if the red marker is found in the frame, it outputs the message, "RedMarker 1." In a similar fashion, the system sends messages pertaining to whether or not the green marker is found in the current frame.

A program is then built to receive these messages being sent to the multicast address. The messages are parsed and one of four counters is updated based on an incoming message: two for located red and green markers and two for red and green markers that were not located.

Five runs of approximately two minutes were carried out, where the markers were moved around the staging area such as they would under normal use of the system. Table 6.1 shows the results of these tests.

It can be seen that the system was able to locate 99.34% of all red markers and 99.29% of the green markers, which is very reasonable.

Total frames with located red marker	5625
Total frames without located red marker	40
Total frames with located green marker	5574
Total frames without located green marker	37
Percentage of red markers found	99.34%
Percentage of green markers found	99.29%

Table 6.1 – Percentage of markers located

6.2 System Accuracy

Accuracy is the second most important non-functional requirement for this system. The accuracy of the 3DHP system is measured with the help of the existing tracking system in the Discovery lab. The current system utilizes an array of ultrasound speakers that emit a sonic chirp. These chirps are detected by several microphones in a handheld wand, which are used to triangulate the location of the wand in 3D space. The system was professionally installed and calibrated to be accurate within fractions of a millimeter.

The 3DHP hand markers are simply taped to this wand. A program is created to poll the wand at a regular interval and send its location to a network address. The 3DHP system is run alongside this program, and the hand marker positions are relayed to the standard network address. A receiver program grabs both sets of hand positions and logs them in a text file.

This method is carried out for both the red and green hand markers, and the results are plotted in Figure 6.3. It can be seen that the hand markers correspond to the locations of the wand with slight differences between the two systems. The hand markers

are taped onto the wand, which could help explain the slight constant difference between the two systems. When the 3DHP markers are moved out from the system origin, the discrepancies become more pronounced. This could be attributed to the distortion introduced by the camera's wide-angle lens. The differences never amount to much more than a few inches, however, which is quite negligible when compared to the dimensions of the stage, which is a 10 foot square. Also, the plots don't show any distinct outliers, which means that we are accurately detecting the desired markers, instead of picking up random objects in the scene. This is very important to the accuracy of the system, as random jumps in the data stream would limit the types of uses of the system.

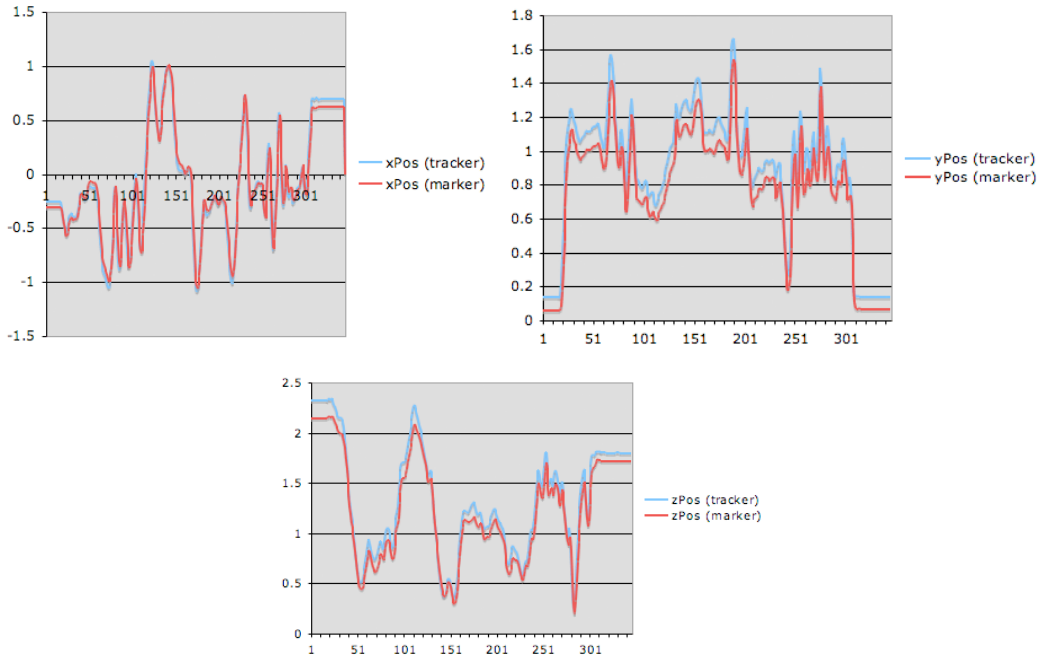


Figure 6.3 – Measured Differences Between 3DHP Markers and Calibrated Tracker for (a) x-axis, (b) y-axis, and (c) z-axis.

6.3 Modifiability

One of the non-functional requirements for the 3DHP system was modifiability. This was addressed by splitting the system functionality into separate modules. By emphasizing separate modules, changes can be made to different aspects of the system without affecting the other pieces. The two main system modules, Image Acquisition and Data Extraction, are implemented in different functions that utilize a shared memory location for passing images. Modifications can be made to the Data Extraction module without needing to change the way the Image Acquisition module is implemented. If we come up with a different method for finding hand positions, we won't have to even touch the current implementation of the Image Acquisition module.

Similarly, if we decide to use a different set of cameras in the system, we only need to be concerned with storing the incoming images from these cameras in the proper memory location. The individual initialization and polling of the cameras is not visible to the Data Extraction module.

6.4 Interoperability

The system was designed to operate alongside the existing tracking system in the Discovery lab. The cameras were mounted above the center screen in a way that they do not interfere with the displays. In addition, since they are mounted above the center screen, the opening and closing of the left and right screens are not affected by the cameras and conversely do not move the cameras from their calibrated positions. Therefore, the system can be used in any of the display modes shown in Figure 5.1.

Since the system uses multicast packets to output found hand positions, multiple systems can use the 3DHP hand positions simultaneously. Applications using the current wand tracking system can access tracking data from its interface while simultaneously grabbing 3DHP hand positions from the network.

6.5 Configurability

The design of the system made use of both a calibration routine and a preferences file, both of which enhance the configurability of this system. The calibration routine can be run independently of the rest of the system, and only needs to be run when camera calibration is desired. Therefore, if the cameras are never moved from their initial poses, the calibration information should stay current and doesn't need to be amended when used in the 3DHP system. However, it is entirely possible to move the cameras to another location – the calibration routine would just have to be run again to determine the new pose of the cameras.

The preferences file allows us to make simple changes to the system without needing to recompile it entirely. The different color ranges to search for or the multicast address and port to send the positions to can be changed by simply modifying this file.

6.6 DAVE

The Digital Audio Visual Environment, otherwise known as DAVE, was created as a proof-of-concept for the 3DHP system – an effort to build a standalone application around the 3D input device. DAVE allows users to manipulate audio samples using standard graphical elements. These elements are manipulated using the 3DHP hand

markers, as opposed to the traditional mouse and keyboard, allowing spectators to visually see the interactions.

6.6.1 DAVE Design

The functionality in DAVE is broken into two individual components – a graphical interface and a sound-processing module. The graphical interface uses hand positions from the 3DHP system to manipulate simple graphical elements on the screen. The various properties of these elements are used to control various properties of different looping sounds in the sound-processing module.

DAVE's graphical interface is based upon three distinct interface elements. The main element in the interface is a draggable box. These boxes are simply two-dimensional squares that can be picked up and dragged around the screen. In addition to the sample boxes, there are two-dimensional "containers," which are simply regions of the screen. The sample boxes can be dragged and dropped onto these container areas. The visual interface also makes use of several buttons, which can be turned on and off. Figure 6.4 shows DAVE's graphical interface and the different elements. The boxes and buttons can be identified as the squares and circles in the screen, respectively. The

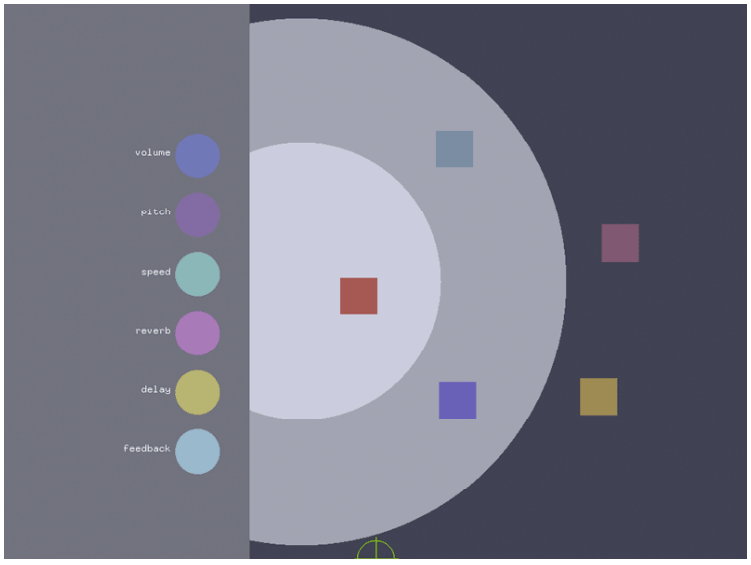


Figure 6.4 – DAVE Graphical Interface

containers are represented by the gray circles in the middle of the screen along with the darker gray background areas of the screen.

These types of graphical elements would normally be interacted with using a computer mouse. A user would be able to click on a draggable box, move it around the screen, and then let go to drop it in a specific location. Similarly, a mouse click would traditionally be used to turn a button on and off.

In order to use the 3DHP system as an input device for DAVE, we can imitate a simple mouse with the three-dimensional hand positions. The x and y hand coordinates can simply be mapped into x and y screen coordinates. The third dimension can be used to emulate a mouse click by setting a certain threshold on the z-axis. A hand marker crossing this threshold can be construed as a mouse button being held down. Likewise, the hand marker crossing back across the threshold can be seen as the mouse button being let back up. Therefore, a simple mouse click would be performed by moving a hand marker across a threshold, such as the plane defined by $z = 0.5$m, and then moving the hand marker back out of this threshold area.

It may seem a bit redundant to use the 3DHP hand coordinates as an input device for DAVE, since it basically emulates a simple mouse. The main benefit to using the 3DHP system is that it gives two hand positions simultaneously. This allows us to work with both hands at the same time; we can drag a box around with one hand while turning a button on or off with the other. DAVE can also be controlled with a standard mouse, giving three different cursors that can interact with the various elements.

An additional benefit of using the 3DHP system as an input device comes from DAVE's usage in performances and demonstrations. Instead of simply moving his

mouse cursor, a performer will be dragging boxes around with his hands on one of the Discovery lab's 10-foot screens. This ability to actually watch what the performer is doing should better maintain interest for spectators.

DAVE takes cues from these graphical elements to manipulate different sounds in real-time with its sound-processing module. This module consists of several sound processing "units", each with a different looping sample. These samples are passed through their own chain of effects with modifiable parameters.

Each sound sample in the sound-processing module is passed through the following effects:

- Volume: The volume of the looping sound sample may be increased or decreased.
- Speed: The speed at which the sample is played back at can be modified, even allowing the sample to be played in reverse.
- Pitch: The pitch of the sample can be scaled up or down a number of octaves.
- Reverb: A simple reverberation effect can be applied to the sample.
- Delay: The sound sample can be delayed a set number of milliseconds and sent into a feedback loop
- Feedback: A percentage of the delayed sample can be fed back into the system.

Each of the buttons in the graphical interface is assigned to one of these effects. When a button is pressed, its corresponding effect parameters can be modified. When released, no changes to the effect parameters are made. The 3DHP system is used to adjust these effect parameters. When an effect button is pressed, its parameters are

changed based on the height of one of the 3DHP markers. When the user moves his hand up (increase in y-position), the effect is increased (increase in parameter values). Similarly, when the hand is lowered (decrease in y-position), the effect is decreased.

How these effect changes apply to the different samples is determined by which container a sample resides in. There are three distinct container areas in the GUI – "inactive/locked," "active/locked," and "active/unlocked." The DAVE interface is tinted red, yellow, and green to outline the locations of these containers in Figure 6.5. Samples placed in the "Inactive/Locked" regions (red tint) will not be played (inactive). In addition, any attempted changes to their effect parameters will be ignored (locked). Samples in the "Active/Locked" region (yellow tint) will continuously play (active), but no changes to their effect parameters will be made (locked). Finally, samples placed in the "Active/Locked" region (green tint), will play (active) and their effect parameters can be modified (unlocked).

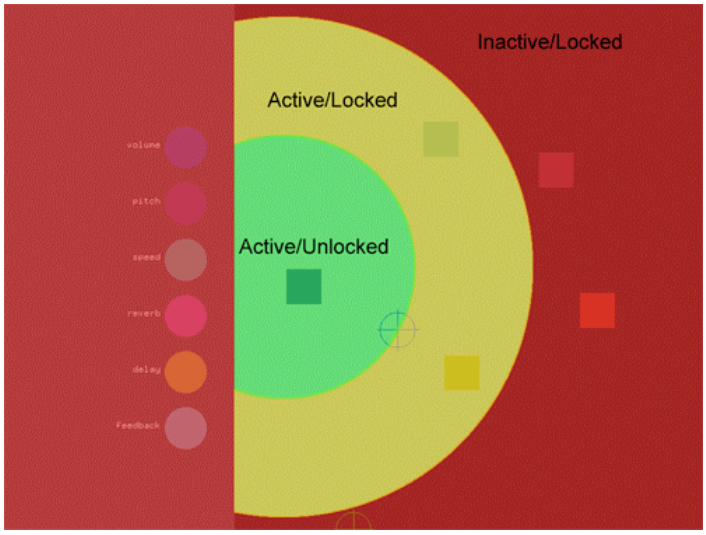


Figure 6.5 – DAVE Regions

6.6.2 DAVE Implementation

DAVE was implemented in the summer of 2005, with the assistance of ARSC student employee Quinton Harris and ARSC summer intern Sean Waite. The

implementation of the two elements (graphical interface and sound processing module) was carried out concurrently.

The graphical interface was implemented in C++ on an Apple G5 Mac Pro. The program was built using GLUT (OpenGL Utility Toolkit), a library of OpenGL utilities. In addition to GLUT, the graphical interface utilizes the Berkeley sockets API to receive the 3DHP multicast packets. Finally, the OpenSoundControl protocol is chosen for sending messages from the graphical interface to the sound-processing module.

OpenSoundControl (OSC) is a communication protocol developed by the UC Berkeley Center for New Music and Audio Technology (CNMAT). This flexible protocol sends messages over a network as UDP packets. The specific implementation used for the graphical interface is oscpack, a set of C++ functions for formatting and sending OSC packets.

Several types of messages are sent from the graphical interface to the sound-processing module. Messages are sent when effect buttons are toggled and when sample boxes are dropped into containers. These message types and example OSC messages are given in Table 6.2.

Message Type	Example	Meaning
Effect-message	/effectButton/volumeButton 0	"volumeButton" has been turned off
	/effectButton/pitchButton 1	"pitchButton" has been turned on
Sample-message	/sample/sample0 0 0	"sample0" has been dropped into the "Inactive/Locked" region
	/sample/sample1 1 0	"sample1" has been dropped into the "Active/Locked" region
	/sample/sample2 1 1	"sample2" has been dropped into the "Active/Unlocked" region

Table 6.2 – Messages sent from graphical interface to sound-processing module

The sound-processing module was developed using the graphical audio development environment Max/MSP. This development environment consists of various objects that can be connected together in a processing chain called a patch. In addition, these patches can be viewed as objects in another patches, giving the application designer a visual representation of object-oriented programming. The Max/MSP implementation of DAVE's sound processing module is built primarily using the set of standard objects included with the base Max/MSP application. These objects include buffers for storing the sound samples, sample playback objects, FFT objects for scaling a sample's pitch, delay objects for synthesizing a reverb and feedback loop, and even a networking object for receiving the 3DHP multicast packets. In addition to the standard objects, the sound-processing module makes use of an external Max/MSP object for receiving the OSC messages from the graphical interface, provided by CNMAT.

The sound-processing module makes use of several individual sample-processing patches. Each of these sample-patches sends a looping sound sample through the chain of effects described previously. Each patch can be locked or unlocked, meaning the changes to the effect parameters will be ignored or applied, respectively. In addition, the playback of the sample can be started or stopped.

The main patch of the sound processing module is responsible for three things: forwarding sample-messages to the sample-patches, updating the effect parameters based on the 3DHP markers, and forwarding these effect parameters to the sample-patches.

The sample-messages tell the sample-patches whether the current sample is supposed to be playing and whether or not to ignore effect parameter changes. This information is simply forwarded to the appropriate sample-patches.

The different effect parameters are changed whenever their respective effect buttons are pressed. When an effect button is pressed, its corresponding effect parameter value will be scaled according to the y-position of the 3DHP marker0. The range of y-position values for marker0 is clipped to a range of 0.5 meters to 1.5 meters – a useable arm height range for the average user. The location of the marker in this range is then mapped to a range of effect parameter values. Table 6.3 outlines the range of values that the effect parameters can be scaled.

Effect Parameter	Low value	High value
Volume	0.0: No volume for sample	1.0: Sample is played back at full volume
Speed	-2.0: Sample is played back at two times the original speed, in reverse	2.0: Sample is played back at two times the original speed
Pitch	-1.0: Sample's pitch is scaled one octave down	1.0: Sample's pitch is scaled one octave up
Reverb	0.0: No reverb assigned to the sample	1.0: Full amount of reverb effect is assigned to the sample
Delay	0.0: No amount of delayed signal is fed back into the audio stream	2000.0: 2,000 milliseconds of the original signal is fed back into the audio stream
Feedback	0.0: None of the fed back signal is sent to the audio stream	0.9: 90% of the fed back signal is sent to the audio stream

Table 6.3 – Effect parameter value ranges

For example, if the Volume button is pressed and marker0's y-position is at 0.5 meters or lower, the volume parameter will be assigned 0.0. If marker0 is raised to a y-position of 1.5 meters or higher, the volume parameter will be assigned 1.0. If marker0's y-position is somewhere in the range of 0.5m – 1.5m, the volume parameter will be scaled linearly from 0.0 – 1.0.

The different effect parameters are forwarded to the sample-patches simultaneously. Each of the effect parameters will then be either ignored by the sample-patch or applied to the desired effect, depending on whether the sample-patch is locked or unlocked.

6.6.3 DAVE Results

Three distinct applications are used in DAVE: the 3DHP system as an input device, the graphical interface, and the sound-processing module. They all combine to create a different way to interact with a user interface and the resulting audio. From the manipulation of boxes by moving hand markers in the air, to the resulting manipulation of the outgoing audio by these hand movements, DAVE serves as an interesting and immersive audio-visual experience.

7. Conclusion

The goal of this project was to create a new 3D input device for the Discovery lab. A method for turning a pair of image coordinates into a single 3D world coordinate was determined, along with methods for calibrating the cameras in the system. The project requirements were outlined, which led to a complete system design and architecture. The system was then implemented, including both the software modules and the various hardware components.

The system was then tested to see how well it followed the various non-functional requirements that were previously outlined. The system performs satisfactorily, both with the total output 3D positions per second (~18.62 per second, Figure 6.1) and the percentage of markers determined in a scene (~99.32%, Table 6.1). The system is also

quite accurate, closely following the precisely calibrated positions from the existing tracking system in the Discovery lab (Figure 6.3).

An entirely new application, DAVE, was specifically created for use with the 3DHP system. DAVE allows a user to manipulate graphical objects and a resulting sound by moving the 3DHP markers in the air – a much different form of interacting with a musical application. The added benefit of two hand markers allows users to be able to control DAVE more precisely; the user is able to turn off an effect with one hand when he has the proper amount desired using his other hand. Most importantly, DAVE shows that the 3DHP system can be used as an additional input device for applications, which was the original goal of the project.

Appendix A – Code Listing

The following code was written for the 3DHP system:

- **3DHP.cpp** – The main program containing the implementations of the Image Acquisition, Data Extraction, and Data Transport modules. This file contains the following functions:
    - main() – thread initialization and main program loop
    - keyboardInput() – thread that checks for keyboard input
    - imageAcquisition() – Image Acquisition module function thread
    - dataExtraction() – Data Extraction module function thread
    - dataTransport() – Data Transport module function
    - parseFile() – Opens and parses the preferences file
    - initializeCameras() – camera initialization
    - initializeNetworking() – networking initialization
    - selectObject() – called by Data Extraction module to select an object based on a specified color range
- **preferences.txt** – Preferences file containing multicast address, multicast port, multicast packet TTL, left marker color range, right marker color range

In addition to the main 3DHP.cpp program file, the following files were created for the camera calibration:

- **cameraCalibration.dev** – The Calibration routine, written for use in HALCON's hDevelop development environment. Reads or creates the following files.
- **CamParametersL.dat** – camera intrinsic parameters for the left camera
- **CamParametersR.dat** – camera intrinsic parameters for the right camera
- **caltab.descr** – calibration plate mark locations
- **caltab.ps** – calibration plate postscript file (for printing plate)
- **relPose.dat** – relative pose information between left and right cameras
- **pose_CCS_to_WCS.dat** – pose information for camera coordinate system

For the DAVE system, the following files were created for the graphical interface:

- **main.cpp** – the main processing thread and GLUT wrapper
- **DaveApp.cpp/DaveApp.h** – the application specifics
- **MVC.cpp/MVC.h** – class for model/view/control programming style
- **BluiOSC.cpp/BluiOSC.h** – OpenSoundControl wrapper
- **Cursor.cpp/Cursor.h** – class for handling 3DHP cursors
- **Clickable.cpp/Clickable.h** – Clickable object class
- **Button.cpp/Button.h** – Button object class
- **EffectButton.cpp/EffectButton.h** – Specific effect button class
- **Draggable.cpp/Draggable.h** – Draggable object class
- **networking.h** – Specific networking functions
- **drawing.h** – Drawing functions

For DAVE's sound-processing module, the following Max/MSP patches were created:

- **DAVESoundProcessing.mxb** – The main sound-processing patch, which makes use of the following subpatches.
- **DAVESample.mxb** – The sound-processing patch for each individual sound sample
- **DAVEChooseSamples.mxb** – patch for assigning sound files to the samples
- **OSCMessages.mxb** – receives and parses the OSC messages from the graphical interface